Calculating the Medial Axis of CAD Models by Multi-CPU based Parallel Computation

Housheng Zhu[1], Yusheng Liu[1]*, Jing Bai[2], Hongwei Wang[3]

[1]State Key Lab. of CAD&CG, Zhejiang University, Hangzhou, P.R. China, 310027
[2]Beifang University of Nationality, Yinchuan, P.R. China, 750021
[3]School of Engineering, University of Portsmouth, Portsmouth, PO1 3DJ, United Kingdom
ysliu@cad.zju.edu.cn

Abstract: Computational efficiency is still a great challenge for the generation of the Medial Axis (MA) for complicated CAD models. Current research mainly focuses on CPU-based MA generation methods. However, most of the methods emphasize using a single CPU, which the highly-efficient methods based on parallel computing are still missing. In this study, a parallel method based on multi-CPU is proposed for the efficient MA generation of CAD models using distance dilation. By dividing the whole model into several parts for which MAs are calculated in parallel and then combined, computational efficiency can be greatly improved -in theory computation time can be reduced nearly *K* times if *K* CPUs are used. Firstly, an adaptive division method is proposed to divide the voxelized model into blocks which have nearly the same number of voxels to balance computational burden. Secondly, the local Euclidean Distance Transform (EDT) is calculated for each block based on the existing distance dilation method. Thirdly, the complete inter-dilation method is proposed to compute the influence between different blocks to get a global EDT for each block. Finally, each block generates a sub-MA separately and then all the generated MAs are combined to obtain the final MA. The last three processes can be efficiently conducted in parallel by using multiple CPUs. Several groups of experiments are conducted which demonstrate the good performance of the proposed methods in terms of effiency.

Keywords: Medial axis; CAD models; Distance dilation; Parallel computing

1. Introduction

The concept of Medial Axis (MA) was first introduced by Blum [1] as a tool for efficient 2D shape description, which has also been extended later on for the description of 3D models. Different from the curve/surface skeleton and the mid-surface, it has a rigid mathematical definition as follows: the MA of an object is the set of all the points having more than one closest point on the object's boundary [2]. It has proven to be useful in many engineering fields, such as finite element analysis, form analysis, path planning in robotics, solid modeling, and mesh generation [3]. Generally, the methods of initial MA generation can be divided into three types, namely the thinning-based methods, the tracing methods and the methods based on the Voronoi graph.

In the thinning-based methods, a model is usually approximated at first. For instance, Nackman put forward a method that substituted a polygon/polyhedron for the smooth boundaries and thus used the MA of the polygon/polyhedron as that of the input model [4]. Ma *et al.* [5] presented a method to approximate MA in which a set of points is sampled from a surface alongside the vectors that are normal to the surface at those points. Xia *et al.* [6] used Cartesian grids and calculated the distance field by solving the hyperbolic-natured Eikonal equation. Gao *et al.* [7] proposed a method based on distance dilation to compute the MA voxels for voxelized models. Those thinning-based methods may not be fast enough, as no parallel computing is employed.

Based on local continuity, the tracing methods in essence generate a MA by tracing some special MA points. The key issue is to determine the crucial bifurcation points and offset the boundary inward. Lee proposed a method which achieves a computation complexity of O ($n$log$n$) for convex polygons and a complexity O (n^2) for concave polygons [8]. For the multiply connected region with h holes, Srinivasan and Nackman developed a method with a computational complexity of O(nh+$n$log$n$) [9]. Recently, Chin provided a linear-time algorithm for computing the MA of a simple polygon [10]. However, it is not a simple task to extend the tracing method for complex models with freeform boundaries [11].

Based on the duality relation between the Voronoi diagram and Delaunay triangulation, the Voronoi graph-based methods calculate a MA from the Voronoi diagram [12]. For instance, Lavender [13] generated Voronoi diagram based on Octree which divides a space into Voronoi zones according to a given resolution. Based on their work, Meijster *et al*. and Hirata made some improvements to develop a parallel method [14-15]. Dey *et al* [16] proposed a hierarchical, simplified and efficient method which is scale and density independent. Smogavec and Zalik [17] proposed an efficient method for generating a MA of polygons based on the Delaunay triangulation of polygons. Giesen *et al* [18] treated the union of inner Voronoi balls in a plane as the MA. However, a lack of parallel processing limits the efficiency of those methods.

To improve computational efficiency, many researchers have proposed similar GPU-based parallel methods [5, 19-22]. However, they can only be used to calculate the approximate skeletons formed by point clouds. In addition, as the density of discrete points is often high, the time complexity may also be quite high. The distance dilation-based method [7] can iteratively generate a MA of high quality with a complexity of O(rn^3) in r dilation rounds and a resolution of n*n*n. However, the time complexity of this method is not satisfactory. The authors have proposed a Double Queues based Distance Dilation (DQDD) algorithm [23] based on existing distance dilation-based methods to improve time complexity from O(rn^3) to O(n^3).

The method proposed in this study uses multiple CPUs, as opposed to a single one as used in most of the methods mentioned above, to generate an initial MA. The hypothesis of this study is that computational efficiency can be improved by processing different blocks in a CAD model in parallel to generate an initial MA. A GPU is not suitable for this parallel methodsince a GPU thread is generally much slower than a CPU thread. Moreover, multi-CPU processors are still widely used and thus it is worth researching how to speed up MA generation based on existing multi-CPU solutions. In the proposed method, the whole model is divided into several blocks and the DQDD algorithm [23] is adapted for the parallel MA generation in each block firstly. Then, the interactions between different blocks are handled and all the local MAs of these blocks are combined to obtain the final MA for the whole model. By using the parallel method, the complexity of distance dilation is improved from O(n^3) to nearly O(n^3/K) with K processors. As such, the main objectives of this study include: (1) to achieve optimal and adaptive division of a CAD model into different blocks with nearly the same number of voxels by considering the balance of computational burden. (2) to efficiently facilitate the interactions between different blocks to obtain the global EDT and on this basis to obtain the final MA.

This paper is organized as follows. Some basic concepts and an introduction to the proposed method are given in Section 2 and Section 3, respectively. The adaptive block division method is discussed in Section 4. The improved distance dilation method for global EDT calculation of the whole part is detailed in Section 5. In Section 6, the generation of a MA using the EDT obtained using the distance dilation method is described. The strategies for computational efficiency improvement are discussed based on a complexity analysis de-

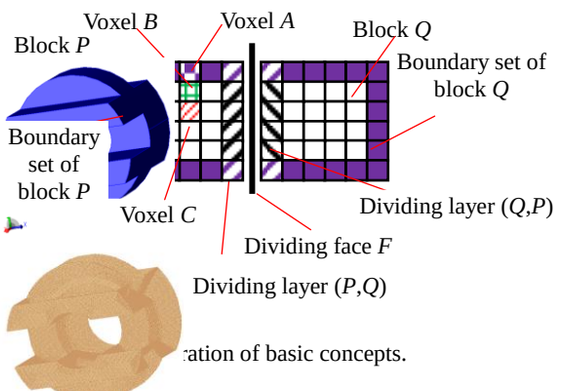
tailed in Section 7. After that, the implementation of the algorithm and its application s to a number of examples are given in Section 8. Finally, the conclusions and future work are discussed in Section 9.

2. Basic concepts

Some frequently used concepts are firstly defined and explained in this section. A 2D model alongside indications of the concepts being used on the model is given in Fig. 1 to assist understanding.

Definition 1: A *block* in a voxelized model refers to an individual part obtained by dividing the model. Here, the *dividing faces*, which are perpendicular to coordinate axis *X*, *Y* or *Z*, are used to divide the bounding box of the model into several blocks. The number of voxels in a block is called the *block size of that block*, and the number of voxels in all blocks is called the *total size*. The *boundary voxel set* (referred to as boundary set throughout this paper) of a block/model refers to all boundary voxels of the block/model. As shown in Fig. 1, a dividing face *F* divides the voxelized model *M* into two blocks, namely *P* and *Q*. The boundary set of blocks *P* and *Q* is composed of several purple boundary voxels.

Definition 2: The *dividing voxel layer* (referred to as dividing layer throughout this paper) is a voxel layer in a block that is adjacent to the dividing face. For two adjacent blocks, *P* and *Q*, divided by a dividing face, dividing layer (*P*, *Q*) is in block *P* and adjacent to block *Q*, and dividing layer (*Q*, *P*) is in block *Q* and adjacent to block *P*. The two dividing layers form a dividing layer pair and have each other as their dual dividing layer. Take Fig. 1 as an example, dividing layer (*Q*, *P*) and dividing layer (*P*, *Q*), both of which are marked in shadow, form a dividing layer pair and belong to blocks *Q* and *P*, respectively.



ration of basic concepts.

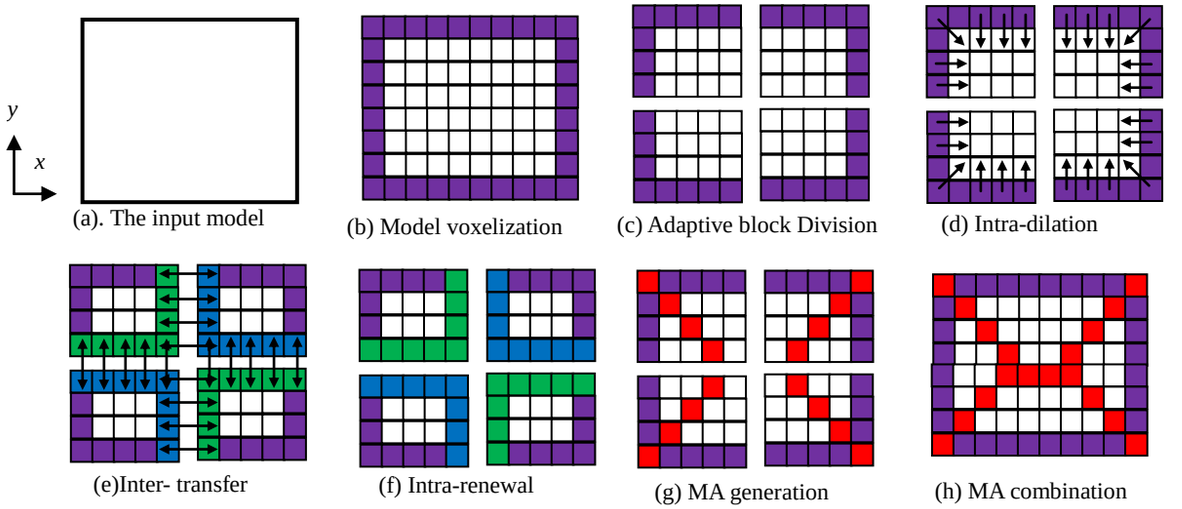
Definition 3: The *distance* of a voxel *A* from a voxel set ***S*** is the Euclidean distance from the center of voxel *A* to the center of the nearest voxel in the boundary set of ***S*** [23]. The corresponding boundary voxel is called a *touch voxel* of voxel *A* from ***S***. Specifically, if ***S*** is a block, the distance is a local distance within that block; and the distance is a global distance when ***S*** is the whole model. The local/global *Euclidean Distance Transformation* (*EDT*) of a block is comprised of the local/global distances of all voxels in the block. As shown in Fig. 1, voxel *A* is the touch voxel of voxel *C* from block *P*. The distance of voxel *A* from block *P* is the distance from the center of voxel *A* to that of voxel *C*.

Definition 4: The *distance dilation* (referred to as dilation throughout this paper) from voxel *A* to its neighbor voxel *B* means updating the distance of voxel *B* using the distance value of voxel *A* [23]. If the touch voxel of voxel *A* can provide a shorterr distance for voxel *B*, the touch voxel becomes a new touch voxel of voxel *B* and the distance of voxel *B* is updated.

As shown in Fig.1, the touch voxel of voxel *A* is itself and the touch voxel of its neighbor voxel *B* is not determined. During the dilation process from voxel *A* to voxel *B*, voxel *A* becomes a touch voxel of voxel *B* since it provides a shorter distance for voxel *B*. The dilation operations in a block can be conducted on some voxels repeatedly until all the voxels in the block have their distances. Those dilation operations from all the voxels in a block is called *complete dilations* from these voxels in the block. As shown in Fig. 1, the dilation operations are conducted from boundary voxel *A* to voxel *B* and then to voxel *C*. Similarly, the dilation operations are conducted from other boundary voxels in block *P*. Finally, all the voxels in block *P* have their distances and these dilation operations are called complete dilations from the boundary set in block *P*.

3. An overview of the MA generation method

To explain the proposed parallel computation method for MA generation, the whole process is described briefly as follows with the help of a 2D model shown in Fig. 2.



(a). The input model (b) Model voxelization (c) Adaptive block Division (d) Intra-dilation

(e)Inter- transfer (f) Intra-renewal (g) MA generation (h) MA combination

Fig. 2. The parallel MA generation process pro-

(i) **Voxelization**: In this step, a CAD model is discretized into a voxelized model. Fig. 2b shows the voxelized model for the input model in Fig. 2a. In Fig. 2b, the purple voxels mean boundary voxels and the white voxels mean body voxels.

(ii) **Adaptive block division**: The voxelized model is automatically divided into several blocks with nearly the same number of voxels by an adaptive division scheme. This is used to balance the computational burden in multi-CPU based parallel computing in the following steps. As shown in Fig. 2c, the model is divided into 4 blocks by two dividing faces that are perpendicular to coordinate axes *X* and *Y*, respectively.

(iii) **Intra-dilation**: The complete dilation the from boundary set within a block are conducted independently. These dilations constitute the intra-dilation of that block by which the local EDT of each block is generated. The arrows shown in Fig. 2d indicate the direction of intra-dilation.

(iv) **Inter-dilation:** This step contains two sub-steps, namely *inter-transfer* and *intra-renewal*. The inter-transfer sub-step contains dilations from the voxels of a dividing layer to its neighbor voxels in the dual dividing layer while all other voxels remain unchanged. As shown in Fig. 2e, the dilation is conducted in four dividing layer pairs in green and blue, and the arrows show the dilation directions. After that, some voxels on the dividing layers update their distances. The complete dilations from those updated voxels on the dividing layers within a block are conducted in the block while voxels in other blocks remain unchanged, which are called the intra-renewal of that block.

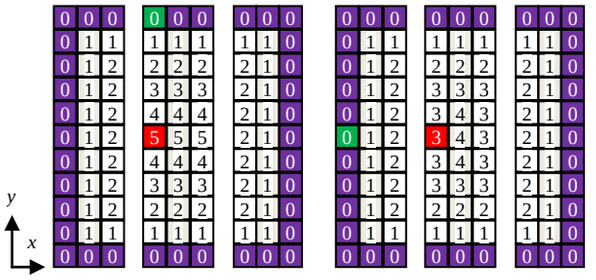
The inter-dilation is conducted between all blocks repeatedly until no distances need to be updated in inter-boundary set transfer or intra-renewal. The complete inter-dilation is comprised of all these inter-dilations. As shown in Fig. 2f, no intra-renewal is further needed and thus the complete inter-dilation ends. Essentially, the inter-dilation is used to consider the interactions between different blocks to obtain the

global EDT for the whole model.

(v) **MA generation and combination**: Firstly, MA voxels are determined for each block in parallel. After that, the voxels on the dividing layers are rechecked to determine whether they are the MA voxels of the model. Finally, the resultant MA of the whole model is obtained by combining the sub-MAs generated for all the blocks. As shown in Fig. 2g, those voxels marked in red are the MA voxels. Fig. 2h shows the resultant MA generated using the proposed parallel method.

As discussed above, distance dilation is an important step in the proposed parallel MA generation method. Initially, it was proposed in the authors' previous work [23] on computing the EDT for a CAD model with only one block and based on a single CPU. In this study, it is improved in two aspects to support multi-CPU based parallel computation of global EDT. Firstly, it is adapted for computing the local EDT of the divided and unclosed blocks while the previous dilation method focused on closed models.

The second improvement is its extension to compute the EDT for a CAD model with a group of blocks in parallel by considering the interactions between these blocks. As shown in Fig.3a, suppose the model is divided into three blocks and 3 CPUs are used to conduct the intra-dilation process in parallel for them separately. Boundary voxels are marked in purple. The local EDTs of these three blocks are computed in Fig.3a. Here, the local distance of the red voxel is 5 and its local touch voxel is marked in green. After the interactions between different blocks are considered by conducting a complete inter-dilation, the distance information of each voxel will be changed, as shown in Fig.3b. The global distance of the red voxel is renewed as 3 by the inter-transfer process which is used to update the global distance information of other voxels for the subsequent intra-renewal process. By using this method, the existing DQDD algorithm is improved for multi-CPU based parallel computation of the global distance of voxels.



(a) Local EDT result (b) Global EDT result

Fig. 3. Local EDT and global EDT

4. Adaptive block division

As mentioned in Section 3, an input CAD model is voxelized first [24]. Then, it is divided into several blocks to conduct intra-dilation for each block in parallel.

Since the maximum number of threads for parallel running is K for a computer with K CPU processors, the entire model is generally divided into k blocks ($k \le K$) to allow the intra-dilations to be implemented by k processors in parallel. Although there are various methods for dividing a model, it is obvious that all parallel threads can finish the intra-dilations in the shortest time if k is equal to K and the sizes of all the blocks are the same. However, it is not a trivial task to ensure that all the blocks have exact same the block size because of irregular and complex model shapes. As such, an adaptive division method is proposed to address this issue.

4.1 Evaluation factors for block division

To evaluate the performances of different block division modes before intra-dilation is conducted, some evaluation factors need to be identified and analysed. The initial block division is proposed firstly.

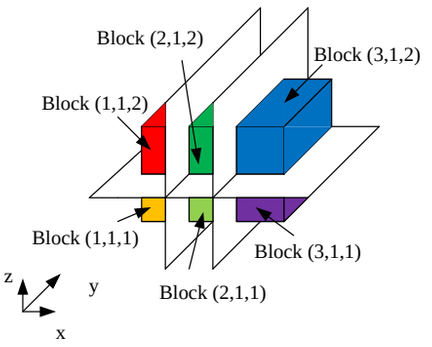


Fig. 4. Block division of initial Div(3,1,2)

Initial block division Div(*div_x*, *div_y*, *div_z*): Suppose there are three dividing faces (*div_x*-1), (*div_y*-1), (*div_z*-1) that are perpendicular to coordinate axes *X*, *Y*, *Z*, respectively for dividing the bounding box of a model equally. The number of the divided blocks is *div_x***div_y***div_z*. This division is called initial Div(*div_x*, *div_y*, *div_z*). Each block is called block (*i*, *j*, *k*) if it is located on the *i*-, *j*-, *k*-th position of the *X*, *Y*, *Z* axes, respectively. As shown in Fig 4, the bounding box of a model is divided into 6 blocks and is denoted as Div (3, 1, 2).

Volume factor: It is defined as the ratio of the block size of the largest block to the average block size, which can be describedusing the following equation (1).

$$\text{Volume factor} = \frac{\max(N_{BS}(x,y,z))*K}{\sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1}N_{BS}(x,y,z)} \quad (1)$$

In the above equation, $N_{BS}(x, y, z)$ is the block size of Block (x, y, z). Suppose a model has four blocks whose sizes are 100, 200, 300 and 400 respectively. As 400 is the largest block size while 250 is the average block size, therefore the volume factor of the model is 1.6.

As $\max(N_{BS}(x,y,z))$ is the largest block size in all *K* blocks, Equations (2) and (3) can be deduced:

$$\max(N_{BS}(x,y,z)) \geq \sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1}N_{BS}(x,y,z)/K \quad (2)$$

$$\max(N_{BS}(x,y,z)) < \sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1}N_{BS}(x,y,z) \quad (3)$$

From equation (1)-(3), it is known that the volume factor for each division mode of the given model is between 1 and *K*. In addition, the value of volume factor is equal to 1 if and only if all blocks have the same block size.

As the degree of parallel computation of the proposed method depends on the total work load and the maximum workload of multiple CPUs, the following equation can be deduced by using Equation (1).

Degree of parallel compuation

$$= \frac{\sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1}N_{BS}(x,y,z)}{\max(N_{BS}(x,y,z))} = \frac{K}{Volume\_factor} \quad (4)$$

It can be seen from Equation (4) that the volume factor can be used to evaluate the balance of block volumes during the block division process.

Edge factor: Suppose the bounding box of a given model is divided into some cuboids with the same size. The edge factor of a block is defined as the length ratio of the longest edge and the shortest edge of its cuboid using Equation (5).

$$\text{Edge factor} = \frac{\max(Edge_x, Edge_y, Edge_z)}{\min(Edge_x, Edge_y, Edge_z)} \quad (5)$$

In Equation (5), $Edge_x$, $Edge_y$ and $Edge_z$ are the edge lengths of the cuboid on *x*, *y*, *z* coordinate axes, respectively. For example, a model is divided into 8 cuboids with the same size. Each cuboid has 10, 20, 30 voxels on *x*, *y*, *z* coordinate axes, respectively. As 30 is the maximum edge length and 10 is the minimum one, the edge factor of the model is 3.

According to Equation (5), the edge factor for each division mode of the given model is not less than 1. It is used to evaluate the interactions between different blocks during inter-dilation and should be minimized.

Renewal factor: This is defined as the ratio of the updated voxels during the complete inter-dilation to all voxels in a block, as described in the the following equation.

$$\text{Renewal factor} = \frac{\sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1} N_{UBS}(x,y,z)}{\sum_{x=0}^{div\_x-1}\sum_{y=0}^{div\_y-1}\sum_{z=0}^{div\_z-1} N_{BS}(x,y,z)} \quad (6)$$

In Euqation (6), $N_{UBS}(x, y, z)$ is the updated block size of Block (x, y, z) by complete inter-dilation. Suppose a model has four blocks whose sizes are 100, 200, 300 and 400, respectively. The updated voxels by complete inter-dilation in these four blocks are then 10, 10, 20 and 20, respectively. As the average block size is 250 and the updated voxels in a block averagely is 15, the renewal factor of the model in this case is 0.06.

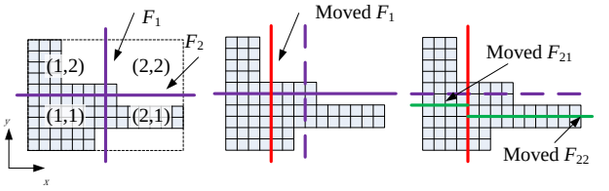
In Equation (6), updated block size of each block is equal to or smaller than the full block size of each block. As a result, the renewal factor for each division mode of a given model is between 0 and 1. This parameter thus can be used to evaluate the cost of complete inter-dilation caused by updating of block sizes.

4.2 Optimal adaptive division

There are many possible initial division modes for a single model when K processors are used. The aim of the adaptive division scheme is to find an optimal division automatically. To obtain the adaptive Div(*div_x, div_y, div_z*) by minimizing edge factor and volume factor, an adaptive division method is proposed in this study which mainly consists of the following two steps:

(1) **Initial division by minimizing edge factor**: the model is divided firstly based on its bounding box in which the bounding box is divided equally. To achieve this for a voxelized model with a resolution of *No_x*No_y* No_z*, the numbers of voxels on the three edges of the blocks are *No_x/div_x*, *No_y/div_y*, *No_z/div_z*, respectively. Here, only the initial division which has minimum volume factor is chosen for the initial division and *No_x*, *No_y*, *No_z* cannot be equal or larger than 3 at the same time.

(2) **Adaptive division by minimizing volume factor**: The positions of *div_x*-1 dividing faces (each divided part is called an *X* part) are changed to satisfy the condition that those *div_x X* parts have nearly the same number of voxels. For each *X* part, the positions of *div_y* dividing faces (each divided part is called a *Y* part) are changed to satisfy the condition that each *Y* part has approximately the same number of voxels. For each *Y* part, the positions of *div_z* dividing faces (each divided part is called a *Z* part) are changed to satisfy the condition that each *Z* part has approximately the same number of voxels.



(a) Simple Div(2,2) (b) Move face F_1 (c) Move face F_2

Fig.5. Illustration of the adaptive Div(2, 2)

After these two steps, the only adaptive division with the minimized volume factor and the minimized edge factor is computed automatically. As shown in Fig. 5, the initial division is achieved by achieving the smallest edge factor (shown in Fig. 5a). In this case, the block sizes of the four blocks are quite different. First, the number of voxels on the left *X* part and the right *X* part of dividing face F_1 are calculated. Here, it is obvious that the former is larger than the latter. Therefore, dividing face F_1 moves left until the number difference of voxels between them is minimized. As shown in Fig. 5b, the red face (in 2D space, it is a line) is moved, dividing face F_1. Then, dividing face F_2 is divided into two dividing faces, namely F_{21} and F_{22}, and for each *X* part, the divided dividing faces F_{21} and F_{22} move to minimize the difference between the block sizes of the upper block and the lower block, as shown in Fig. 5c. After the adaptive block size adjustment, the following approximate equations will be met for adaptive Div(2,2):

$$\text{Min}\{|N_{BS}(1,1)+ N_{BS}(1,2)-[N_{BS}(2,1)+ N_{BS}(2,2)]|\} \quad (7a)$$

$$\text{Min}[|N_{BS}(1,1) - N_{BS}(1,2)|] \quad (7b)$$

$$\text{Min}[|N_{BS}(2,1) - N_{BS}(2,2)|] \quad (7c)$$

As a result, the volume factor is minimized to nearly 1. This adaptive division has the minimum edge factor and the minimum volume factor among all possible division modes, and thus can in theory achieve the best per-

formance of parallel computing. The detail proof of optimal adaptive division will be given in Section 7.2.

5. Distance dilation of the Whole Model

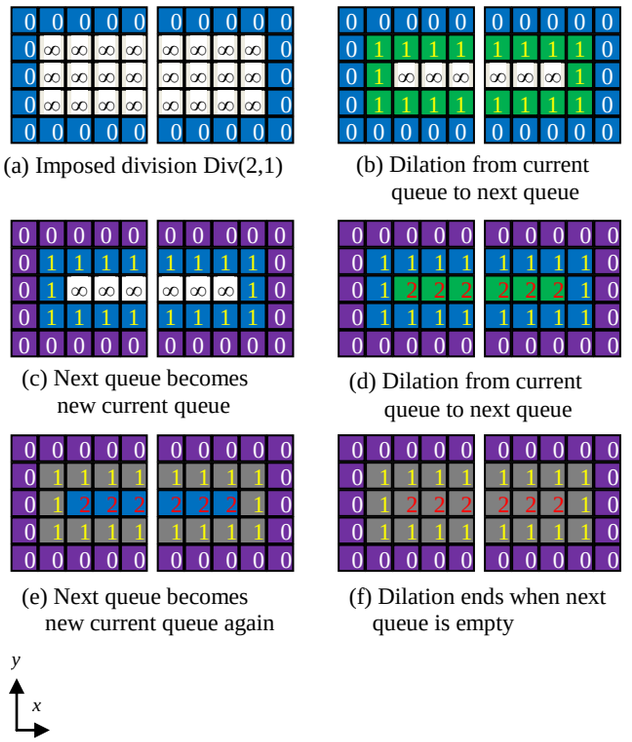
After the model is divided into blocks, the intra-dilation and complete inter-dilation processes need to be conducted for a single block and block pairs in parallel.

5.1 Intra-dilation based on the DQDD algorithm

The detailed DQDD algorithm has published elsewhere and out the scope of this paper [23]. This algorithm uses the *current queue* and the *next queue* which are in the *current range* and the *next range* to conduct complete dilations from boundary set in a block. In this paper, only a brief introduction is givento explain the whole process shown in Fig. 6. In the figure, the current queue is marked in blue; the next queue is marked in green; boundary voxels are marked in purple; and computed voxels are marked in grey. It is noteworthy that the initial distance of the internal voxels is set as infinity instead of zero. Take Block(1,1) in Fig. 6 as an example, set both the queues that will be used in the DQDD algorithm firstly. Here, the current queue is set with the boundary voxels of Block(1,1) in current range and the next queue is set as NULL. The intra-dilation process is described as follows:

(1) For each voxel *A* in the current queue (blue voxels in Fig. 6a), take it out of the current queue, update the distance of its 26 neighbor body voxels by dilation from voxel *A* to each of them, as shown in Fig. 6b. If the distance of any neighbor voxel is updated and the voxel is in next range, push it into the next queue (green voxels in Fig.6b).

(2) If the next queue is not empty, set it to be the new current queue; then, the previous empty current queue is used as the new next queue and another round of dilation begins by updating current range and next range (Fig. 6c); go to Step (1) (Fig. 6d-e). Otherwise, the procedure ends if the next queue is empty (Fig. 6f).



(a) Imposed division Div(2,1) (b) Dilation from current queue to next queue

(c) Next queue becomes new current queue (d) Dilation from current queue to next queue

(e) Next queue becomes new current queue again (f) Dilation ends when next queue is empty

Fig. 6. The intra-dilation process (the current queue is blue; the next queue is green)

After these steps, local distances are correctly obtained for all the voxels of the block, and the local EDT of the block is calculated. With the help of the updated distance by using double queues, the boundary set is dilated as touch voxels from the boundary to the inner part of the block.

As mentioned above, the DQDD algorithm is conducted within the given block and is thus not affected by the other blocks. Therefore, the DQDD algorithm can be executed for each block independently without influencing other blocks and can thus be conducted in parallel for all the blocks.

5.2 Complete Inter-dilation

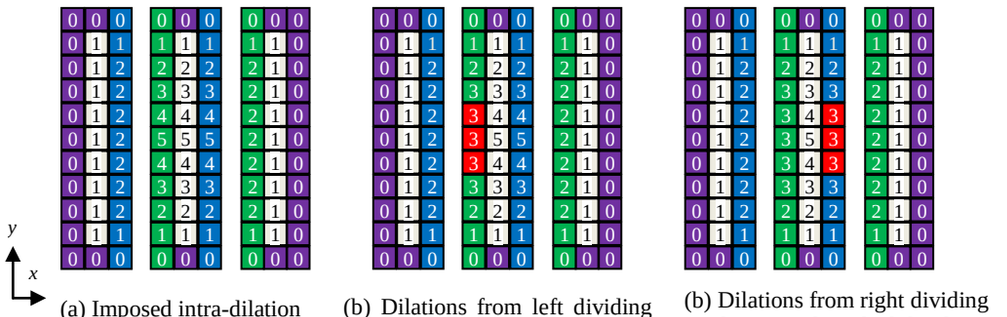
As mentioned above, only the interactions within a block is considered during the intra-dilation process. To compute the global EDT for a given model, interactions between different blocks must be considered as well. Inter-dilation is used to compute the inter-block interactions by distance dilation, which contains two steps: inter- transfer and intra-renewal. Specifically, the former

is used to enable the voxels on a dividing layer to obtain distance information from other blocks. The latter is used to renew the distance information of other voxels using the distance information of the voxels on the dilation layer.

5.2.1 Parallel inter-transfer in the dividing layer pair

For two adjacent blocks, the direct connecting part is the dividing layer pair that is adjacent to a dividing face. To obtain the touch voxels for a given block from its neighbor block, it is necessary to perform dilations between each dividing layer pair. Suppose that dividing layer(*P*, *Q*) and dividing layer(*Q*, *P*) are a dividing layer pair that belong to two adjacent blocks *P* and *Q*, respectively. To conduct inter-transfer between blocks *P* and *Q*, two inter-transfers between dividing layer(*P*, *Q*) and dividing layer(*Q*, *P*) are performed as follows:

[1] For any voxel on dividing layer(*P*, *Q*), conduct the dilation from it to its 9 neighbor voxels on dividing layer(*Q*, *P*).



(a) Imposed intra-dilation (b) Dilations from left dividing layers to their dual dividing (b) Dilations from right dividing layers to their dual dividing lavers

Fig. 7. Illustration of dilation in dividing layer pairs

[2] For any voxel on dividing layer*(Q*, *P*), conduct the dilation from itself to its 9 neighbor voxels on dividing layer(*P*, *Q*). This is in essence reverse operation of Step 1.

Fig. 7b and Fig. 7c illustrate the above two steps of the dilation process in an inter-transfer. Fig. 7a is the result of intra-dilation for the three divided blocks. In the figure, the blue voxels and the green voxels are voxels on two dividing layers of a dividing layer pair, respectively, and the purple voxels are the boundary voxels. The dilation process is conducted between the voxels of each dividing layer and their neighbor voxels on the corresponding dividing layer. As shown in Fig. 7b, dilations are conducted from the blue voxels to the green voxels, and the red voxels are the updated voxels. Then, a dilation process similar to Fig. 7b is carried out in a reverse direction. As shown in Fig. 7c, dilations are conducted from the green voxels to the blue voxels, and the red voxels are the updated voxels in inter- transfer.

Moreover, a parallel computing strategy is also considered here to speed up the inter-transfer process. With the consideration that inter-transfers for different dividing layer pairs do not influence each other, every inter-transfer process on dividing layer pairs that are generated by the same dividing face can use one thread. Because there are *div_x*+*div_y*+ *div_z*-3 dividing faces, the number of threads needed for a parallel inter-transfer is also the same, which is smaller than the block number *div_x***div_y***div_z*. Therefore, there are enough processors for implementing parallel the inter-transfer processes.

5.2.2 Improved DQDD algorithm for parallel intra-renewal

The intra-renewal process can be regarded as a special dilation operation from the updated voxels on the dividing layers. Compared with the general intra-dilation process, there are two characteristics for the intra-renewal process:

[1] The updated voxels on dividing layers of a block have touch voxels from the boundary set of other

blocks in the previous inter-transfer of the same inter-dilation. Those voxels are not in the initial current range.

[2] Each voxel of the block has already had its local distance. Its distance only needs to be updated by using voxels on dividing layers for which distances are updated by a previous inter-transfer of the same inter-dilation.

Due to these characteristics, the DQDD algorithm needs to be extended for the intra-renewal after the following improvements are made.

(1) The updated voxels on dividing layers are set as the initial current queue of intra-renewal since the boundary sets of other blocks may only exist as touch voxels of voxels on dividing layers, which are calculated by previous inter-transfers. Thus, current queue and next queue are not required to be in current range and next range.

(2) Only the voxels that are affected by interactions between blocks will be involved in intra-renewal. This makes the computation time of intra-renewal much shorter than intra-dilation.

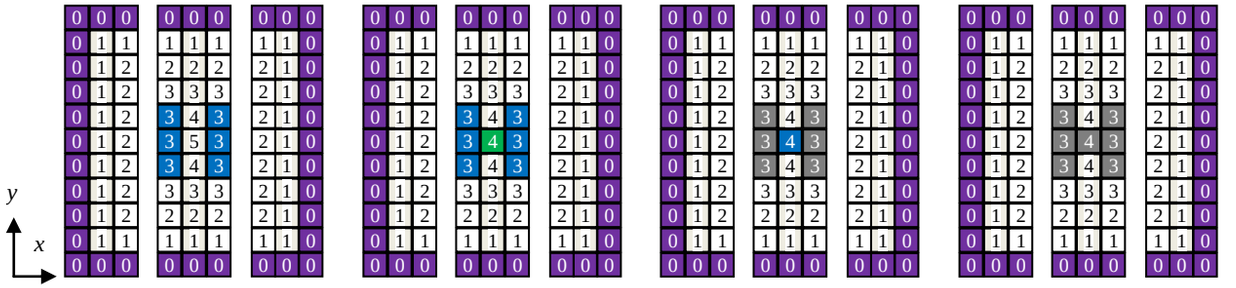
The intra-renewal process is explained using the 2D blocks in Fig. 8. In the fiture, the current queue is marked in blue; the next queue is marked in green; boundary voxels are marked in purple; and computed voxels are marked in grey. Here, the intra-dilation and one round of inter-transfer have been conducted for the model, and the updated voxels by the previous inter-transfer are computed using the method shown in Fig.7. These voxels are pushed into the initial current queue in blue in Fig. 8a. And then the extended DQDD algorithm is implemented as follows after further improvements are made.

(1) For each voxel *A* in the current queue (blue voxels in Fig. 8a), take it out of the current queue. If voxel A is in current range, it updates the distance of its 26 neighbor body voxels by dilation from voxel *A* to each of them. If any neighbor voxel is updated and not queued, push it into the next queue (green voxels in Fig.8b). Else if voxel *A* is not in the current range, push it into the next queue.

(2) If the next queue is not empty, set it to be the new current queue; then, the previous empty current queue is used as the new next queue and another dilation round begins by updating the current range (Fig. 8c); go to Step (1). Otherwise, the procedure ends if the next queue is empty (Fig. 8d).

It can be observed from Fig. 8 that the computational complexity of intra-renewal is much lower than that of intra-dilation. The reason is that during intra-renewal, the distances of most voxels cannot be updated. As a result, they do not need to be dilated.

Moreover, it can be observed that during the intra-renewal of each block, only voxels of the block are involved. Therefore, similar to intra-dilation, it can also be conducted in parallel for different blocks.



(a) Initialize current queue (b) Dilation from current queue to next queue (c) Next queue becomes new current queue (d) Dilation ends when next queue is empty

Fig. 8. Illustration of the intra-renewal process (the current queue is blue; the next queue is green)

5.3 Analysis of inter-dilation times in complete inter-dilation

The aim of complete inter-dilation is to obtain the global distance values for all the voxels. Although one round of inter-dilation is enough to obtain it in most cases, the theoretical maximum rounds of inter-dilation required for complete inter-dilation should be estimated.

5.3.1 Analysis of the maximum times of inter-dilation

If a block P obtains its local EDT from block $B_1,B_2,\ldots B_k$, then $B_1,B_2,\ldots B_k$ are called the *known blocks* of block P. During the intra-dilation of block P, every voxel A of block P obtains its local distance and block P obtains its local EDT. Therefore, block P is a known block for itself. During one round of inter-transfer of block P, the known blocks of the face-neighbor blocks of block P are also known by the dividing layers of block P by dilations from the dual dividing layers. During one round of intra-renewal, the known blocks of face-neighbor blocks of block P are known by block P by the complete dilations in block P. Therefore, one round of inter-dilation will result in the known blocks of the face-neighbor blocks of block P being added to the known block set of block P. In Fig. 9a-d, the known block set of each block after intra-dilation, and the three inter-dilations are shown on the corresponding block.

Therefore, after $|a\text{-}d|+|b\text{-}e|+|c\text{-}f|$ times of inter-dilation, the known blocks of block (d, e, f) will be added to the known block set of block(a, b, c). Here, $1\le a, d \le div\_x$, $1\le b, e\le div\_y$ and $1\le c, f \le div\_z$.

Based on the above analysis, the maximum times of inter-dilations in a complete inter-dilation is $div\_x+div\_y+div\_z-3$ to ensure that any block gets EDT from all the blocks. After that, the EDT of any block is the global EDT, and the inter-dilation ends. As shown in Fig. 9a, a

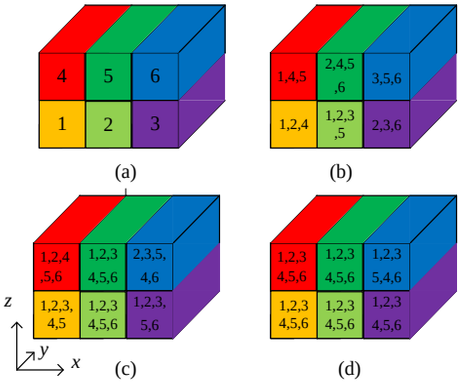


Fig. 9. Known block set of blocks after each inter-dilation

model is divided into six blocks by Div(3,1,2). During four inter-dilations, the known blocks of each block is marked on it as shown in Fig. 9a-d. In this way, after 4 rounds of inter-dilation, each block has got the information about the boundary set of all the blocks.

5.3.2 Optimization of complete inter-dilation

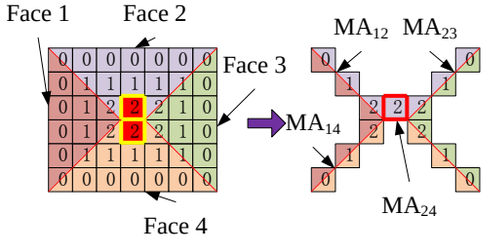
Suppose model M has k blocks (B_1, $B_2,\ldots,B_k$). After $div\_x+div\_y+div\_z-3$ rounds of inter-dilations, the inter-dilation process ends. As $div\_x+div\_y+div\_z-3$ is not a small number, the executed number of inter-dilations in complete inter-dilation needs to be reduced. Here, the boundary set of a block B_i cannot contain the touch voxels of voxels in other blocks if all voxels of block B_i

stop dilation during complete inter-dilation. Obviously, voxels of block B_i cannot dilate to more blocks in this case.

Therefore, if voxels of all blocks stop dilation during the inter-transfer or intra-renewal processes, the complete inter-dilation will end, even though the inter-dilation number is not the maximum. This criterion is checked after each inter-transfer and intra-renewal process in complete inter-dilation for early termination. It will be observed in later experiments that most inter-dilation runs through only one round.

6. MA Generation and Refinement

After the intra-dilation and complete inter-dilation, the global distance is obtained for all the voxels of the input model. The following task is to find the correct MA voxels by using the distance information and refine the result. In this study, the dilation-based method is adapted to calculate the MA voxels [23]. As shown in Fig. 10a, the dilation region is generated first based on the distance of each voxel. Here, the regions with different colors are different regions. It is obvious that the voxels that lie on the intersection of two different dilation regions and on two different boundary surfaces are MA voxels, shown as voxels with two colors in Fig. 10b. The reason is that these voxels have the same distance to two boundaries.



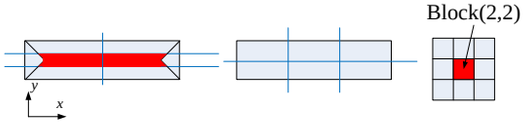
(a) Dilation regions (b) MA voxels

Fig. 10. General process of MA voxel determination

Note in Fig. 10a that only one voxel with a yellow boundary is selected as an MA voxel, shown as the voxel with a red boundary in Fig. 10b. This is the typical problem that occurs for the voxelization-based method, and it is called the double-layer problem. The solution for this problem is that only one layer of voxels is selected as the approximate MA voxels on the condition that the connectivity between the MA voxels must be kept.

To generate MA in parallel, MA generation is conducted in each block in parallel. However, double-layer problem may occur between different blocks. Suppose the two red voxels in Fig.10a are in different blocks, both of them will be considered as MA voxels. To refine MA, additional MA refinement has to be implemented for boundary layer pairs.

The authors proved that the maximum MA error of the dilation-based method is $\sqrt{3}$ **voxel size*/2 and stated some special conditions in MA generation [23]. As a result, the resultant MA of the proposed method is converged when *voxel size* is small enough and its quality is ensured.



(a) Div (2,3) (b) Div(3,2) (c) Div(3,3)

Fig. 11 Renewal factors by different block divisions

7. Computational Efficiency Improvement

It is obvious that the computational efficiency of the proposed parallel MA generation algorithm depends on the intra-dilation and the complete inter-dilation process. Different block division strategies will affect the computational efficiency greatly. Here, the computational complexity of the proposed method is analyzed first, and then some strategies are proposed to improve the computational efficiency as much as possible.

Suppose that the bounding box of the input model has n voxels on the three edges and that the total number of voxels of the model is n^3. Additionally, there are K threads used in parallel, and the model is divided into K blocks.

7.1 Complexity analysis

The total time cost for each MA generation step of a CAD model is described as follows by using parallelism

degree in equation (4). Here, the time for voxelization is not considered for comparison because it is a common part for the voxelization method. Here *vf* and *rf* mean volume factor and renewal factor respectively.

[1] **Time cost for intra-dilation**: Suppose that the time for dilation for one voxel is T_d. The cost time for intra-dilation is $vf*T_d*n^3/K$.

[2] **Time cost for inter-transfer**: dividing layer pairs that are generated by a dividing face have $2*n^2$ voxels. With the consideration that inter-transfer for dividing layer pairs of different dividing faces are conducted in parallel, the time used for one round of parallel inter-transfer is T_d*2*n^2. N_{IT} is the number of necessary inter-transfers, which is equal to or smaller than $div\_x+div\_y+div\_z-3$.

[3] **Time cost for intra-renewal**: Suppose that the average number of voxels that need to be dilated in the intra-renewal of each block is $rf*n^3/K$. The time cost is $vf*rf*T_d*n^3/K$ when *K* threads are used in parallel. N_{IntraR} is the number of required inter-transfers, which is equal or smaller than $div\_x+div\_y+div\_z-3$.

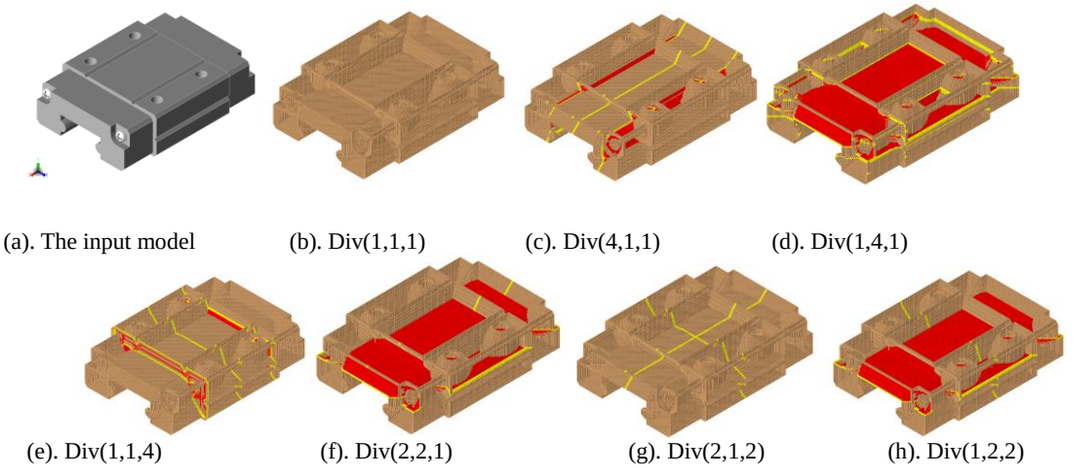
[4] **Time cost for MA generation**: Suppose that MA generation for a voxel costs time T_r. Because all voxels need to be checked, the time cost is $vf*T_r*n^3/K$ when *K* threads are used in parallel.

[5] **Time cost for additional MA refinement**: Suppose that MA refinement for a voxel costs time T_r. There are $2*(div\_x+div\_y+div\_z)*n^2$ voxels on dividing layers to be checked sequentially. Therefore, the time cost is nearly $2*(div\_x+div\_y+div\_z)*n^2*T_r$.

It can be observed that the total time cost of this proposed method is $T_d*n^3 + T_r*n^3$ when only one thread is used, whereas that of the proposed parallel method is $T_d*(vf*n^3/K+N_{IT}*n^2+N_{IntraR}*vf*rf*n^3/K)+ T_r*(vf* n^3/K + 2*(div\_x+div\_y+div\_z)*n^2)$ when *K* threads are used in parallel. Therefore, the proposed method can dramatically reduce the time complexity. The time cost of intra-dilation and MA generation is only vf/K times that of using a single thread which is $O(n^3)$. As *K* is limited by the processor number, reducing the volume factor and complete inter-dilation is imperative to speed up the method.

7.2 Complexity reduction by adaptive division

As mentioned above, the computational efficiency of the proposed method only depends on N_{IT}, N_{IntraR}, renewal factor and volume factor. Here N_{IT}, N_{IntraR} and renewal factor are in turn determined by the boundary set dependency between different blocks. For example, if most of the touch voxels of the voxels in each block are in the same block, N_{IT} and N_{IntraR} will be 0 or 1, while renewal factor will be near to 0. However, N_{IT}, N_{IntraR} and renewal factor will increase if most of the touch voxels of the voxel are in different blocks. At this time, renewal factor



(a). The input model (b). Div(1,1,1) (c). Div(4,1,1) (d). Div(1,4,1)

(e). Div(1,1,4) (f). Div(2,2,1) (g). Div(2,1,2) (h). Div(1,2,2)

Fig. 12 The illustration of MA generation of different division modes

will be near 1 because most voxels need to update their distance using complete inter-dilation. Moreover, N_{IT} and N_{IntraR} will also be large if the block number between a voxel and its touch voxels is large.

Based on the above analysis, the following strategies are used in this study to reduce boundary set dependency between different blocks.

(1) Minimizing edge factor to minimize N_{IT} , N_{IntraR} and renewal factor. As in Div(2,3), as shown in Fig. 11a, the renewal factor is nearly 1 and N_{IT} and N_{IntraR} are 2 and 1, respectively. In Fig. 11b, the division is Div(3,2), the renewal factor is reduced to 0, and N_{IT} and N_{IntraR} are 1 and 0, respectively. Although both Div(2,3) and Div(3,2) contain six blocks, Div(3,2) has a smaller edge factor and its complete inter-dilation costs is less.

(2) The values of *div_x*, *div_y* and *div_z* cannot be equal to or larger than 3 at the same time for the block division. If they are, at least one block has no boundary set. Therefore, the block does not have a local EDT in intra-dilation, and its renewal factor will be 1 in the subsequent intra-renewal. As shown in Fig. 11c, a rectangle (2D) is divided by Div(3, 3). The red block(2, 2) does not contain boundary set and its renewal factor is 1. Obviously, such a block division will heavily affect the efficiency of complete inter-dilation and needs to be avoided.

In subsection 4.2, the adaptive division chooses the division which has minimum *ef* and ensures the values of *div_x*, *div_y* and *div_z* cannot be equal to or larger than 3 at the same time. Therefore, N_{IT}, N_{IntraR}, *rf* are minimized. What's more, the adaptive division minimizes *vf* by moving dividing faces. Finally, the adaptive division offers the minimum complexity. Experiments in subsection 8.1 below give the similar conclusion by comparing the adaptive division and other possible divisions.

8. Implementation and Examples

The proposed parallel MA generation method has been implemented with visual studio 2008 and the geometric modeling kernel ACIS R20 as well as HOOPS 17.00 for visualization. The computer that was used uses an Intel Core i7 3770 with 4 CPU processors. To demonstrate the efficacy and efficiency of the proposed method, three groups of experiments were conducted as follows. The first group is used to compare the time cost of the proposed method for all possible division modes with all 4 CPU processors. Meanwhile the influence of the volume factor and edge factor are also analyzed. The second group is used to compare the time cost of the proposed method for different numbers of threads. Four examples were given in the third group to illustrate the performance of the proposed method using all 4 CPU processors. To unify the examples, all models were voxelized

Table 1. Execution time for the example in Fig. 13 (ms)

division	Time for intra-dilation	Time for CID	Time for MA generation	Time for additional MA generation	Total time	Speedup times
(1,1,1)	1015	---	125	---	1140	---
(4,1,1)	327	32	47	0	406	2.81
(1,4,1)	374	188	47	16	625	1.82
(1,1,4)	312	15	32	15	374	3.05
(2,2,1)	328	62	47	16	453	2.52
(2,1,2)	312	0	47	0	359	3.18
(1,2,2)	312	78	46	0	436	2.61

Table 2. The volume factors and edge factors for the example in Fig. 13

division	Block1 voxel No.	Block2 voxel No.	Block3 voxel No.	Block4 voxel No.	volume factor	*X* axis voxel No.	*Y* axis voxel No.	*Z* axis voxel No.	edge factor
(4,1,1)	264159	174090	184078	243257	1.221	33	51	182	5.52
(1,4,1)	185039	188113	289952	202480	1.340	131	13	182	14.00
(1,1,4)	202795	233415	238678	190696	1.103	131	51	46	2.85
(2,2,1)	187162	251087	185990	241345	1.160	66	21	182	7.00
(2,1,2)	220842	217407	215368	211967	1.021	66	51	91	1.78
(1,2,2)	187772	185380	248438	243994	1.148	131	26	91	5.04

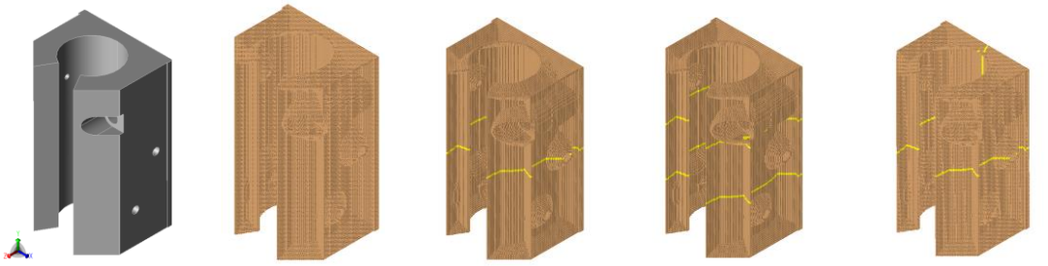
with a resolution of 180 in the longest direction. Here, the MA voxels on boundary layers are colored with yellow to discriminate them from other MA voxels. Red voxels are the MA voxels newly generated during the complete inter-dilation to discriminate them from the brown MA voxels that are generated during intra-dilation.

8.1 Time comparison of different division modes

In this experiment, the model as shown in Fig. 12a is used to conduct time comparison of all possible block division modes. Without loss of generality, all four processors are used in the experiment. All the division modes which divide the bounding box equally with four processors are given in Fig. 12b-h. The time cost of each division mode is tabulated in Table 1; the edge factors and volume factors of each division mode are tabulated in Table 2.

It can be observed from Table 1 that compared with the method with single-thread, the performance of the proposed parallel MA generation method is 1.5-3 times that when 4 processors are used. Meanwhile, it can also be observed that the computational time of different division modes is quite different, even though the same number of processors is used. The reason is that they have different volume factors and edge factors. As shown in Table 2, Div(2,1,2) has the smallest volume factor and edge factor, whereas Div(1,4,1) has the largest volume factor and edge factor. As a result, Div(2,1,2) is the adaptive division here by considering volume factor and edge factor. Table 1 illustrates that Div(2,1,2) actually provides the best speedup performance while Div(1,4,1) provides the worst. As proved in subsection 7.2, this group of experiment illustrates that the adaptive division which has the minimum volume factor and minimum edge factor is the optimal division.

8.2 Time comparison of different numbers of threads



(a). The input model (b). Div(1,1,1) (c). Div(1,2,1) (d). Div(1,3,1) (e). Div(2,2,1)

Fig. 13 The illustration of MA generation for different numbers of threads by adaptive divisions

In this experiment, the time cost of different numbers of threads is compared with an example, as shown in Fig. 13. Based on the analysis in section 7, the adaptive block division of each number of threads is used for comparison, and the results are given in Fig. 13c-e. Table 3 shows the time cost by different division modes. It can be observed that less time is spent when more threads are used. It can also be observed from Table 3 that the time for complete inter-dilation is much less than that of intra-dilation, which ensures a high efficiency of the proposed method with multi-threads. Meanwhile, as the adaptive division is used as the optimal division, the speedup performance is maximized.

8.3 Examples by adaptive divisions with all CPU processors

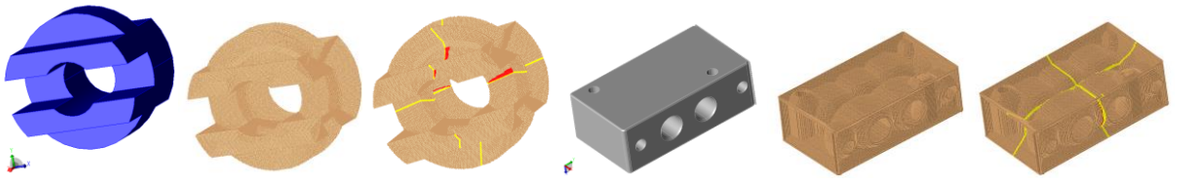
To further illustrate the performance of the proposed method by adaptive division, four different models are used, as shown in Fig. 14-15. Their medial axes are generated by adaptive division with 4 processors to achieve the best performance. Their speedups are tabulated in

Table 3 Execution time for the model in Fig. 13 (ms)

Result No.	Number of Processor	Division	Time for intra-dilation	Time for complete inter-dilation	Time for MA generation	Time for additional MA generation	Total time	Speedup times
1	1	(1,1,1)	1280	---	187	---	1467	---
2	2	(1,2,1)	688	0	93	0	781	1.88
3	3	(1,3,1)	484	15	78	0	577	2.54
4	4	(2,2,1)	390	0	78	0	468	3.13

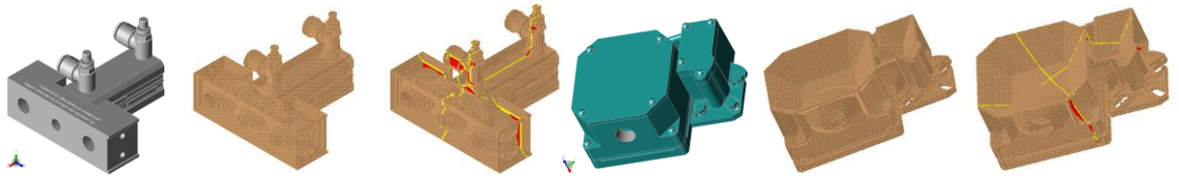
Table 4 Execution time for the examples shown in Fig. 14~17 (ms)

Models	Division	Time for intra-dilation	Time for complete inter-dilation	Time for MA generation	Time for additional MA generation	Total time	Speedup times
Model in Fig. 14	(1,1,1)	2568		382		2950	---
	(2,2,1)	779	14	136	5	934	3.16
Model in Fig. 15	(1,1,1)	1093	---	140	---	1233	---
	(1,2,2)	328	15	47	0	390	3.16
Model in Fig. 16	(1,1,1)	1466	---	202	---	1668	
	(2,1,2)	468	16	78	0	562	2.97
Model in Fig. 17	(1,1,1)	1404	---	202	---	1606	---
	(2,2,1)	437	16	63	0	516	3.11



(a). The input model (b). Div(1,1,1) (c). Div(2,1,2) (a). The input model (b). Div(1,1,1) (c). Div(1,2,2)

Fig.14 MA generation of a sample model Fig.15 MA generation of a sample model



(a). The input model (b). Div(1,1,1) (c) Div(2,1,2) (a). The input model (b). Div(1,1,1) (c) Div(2,2,1)

Fig.16 MA generation of a sample model Fig.17 MA generation of a sample model

Table 4. Here, the shapes of the models in Fig. 14-15 are more symmetrical than those in Fig. 16-17. As a result, the speedups of models in Fig.14-15 are higher than that of models in Fig.16-17. Some other models are also tested in the experiments, with similar results obtained.

9. Conclusions and future work

Recently, some studies have been conducted on the parallel generation of skeletons to improve computational efficiency. All of these studies are devised based on GPU processors. However, they may not be satisfactory for the generation of MAs for CAD models since the interactions between different blocks is too complex when calculations for multiple MAs are conducted in parallel. In addition, a parallel method is needed for CPU-based systems since there are a wide range of applications for these systems. Therefore, this study aims to address the research gap by developing a parallel method on the CPU platform. The contribution of the study is concluded as follows:

(1) An approach to generating MA in parallel using multiple CPUs is proposed, which enables the main steps of intra-dilation, inter-transfer, intra-renewal, MA generation and refinement to be conducted in parallel for different parts. Computational efficiency can be greatly improved compared to existing methods that use only one thread.

(2) An adaptive block division method is proposed to maximize the degree of parallel computing in MA generation by minimizing both the volume factor and the edge factor while reducing the time complexity of the method.

(3) The distance dilation method is adapted and improved to implement obtaining the global distance for all the voxels by using the boundary sets of other blocks and reusing the local distance calculated within a block Meanwhile, the homotopy is also preserved even if the genus is destroyed during the division of the model into several parts.

Future work will be focused on three areas. The first will be on conducting a performance test with more CPU processors. The second involves analyzing the influence of voxel size and in particular multi-scale voxelization will be considered to satisfy the requirement of homotopy in some extreme cases. The third regards more applications of the proposed method. For example, applications of the proposed method in efficient 3D model retrieval and CAD/CAE integration will be considered.

Acknowledgements

The authors appreciate the financial supports from the NSF of China (Grant Nos. 61173126 and 61163016) and the 863 High-Technology Project of China (Grant No. 2011AA100804).

References

[1] H. Blum. A transformation for extracting new descriptors of shape. In: Models for the Perception of Speech and Visual Form, 1976: 362–380.

[2] http://en.wikipedia.org/wiki/Medial_axis

[3] F. Frederic, Leymarie and B. Benjamin Kimia. The medial scaffold of 3d unorganized point clouds. IEEE Trans. Pattern Anal. Mach. Intell., 29(2):313-330, 2007.

[4] L. R. Nackman. Curvature relations in three-dimensional symmetric axes. Computer Graphics and Image Processing, 1982, 20: 43-57.

[5] J. Ma, S.W. Bae and S. Choi. 3D medial axis point approximation using nearest neighbors and the normal field. Visual computer 28(1):7-19, 2012.

[6] H. Xia, P.G. Tucker Fast equal and biased distance fields for medial axis transform with meshing in mind. Applied mathematical modeling 35(12): 5804-5819, 2011

[7] W. Gao, S.M.Gao, Y.S. Liu. Multi-resolutional similarity assessment and retrieval of solid models based on DBMS. Computer Aided Design, 2006: 38：985-1001.

[8] D. T. Lee. MA transformation of a planar shape. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1982, 4(4): 363-369.

[9] V. Srinivasan and L. R. Nackman. Voronoi diagram for multiply connect polygonal domains, I: Algorithm. IBM Journal of Research and Development, 1987, 31(3): 361-372.

[10] F. Chin, J. Snoeyink, C.A. Wang. Finding the medial axis

of a simple polygon in linear time. Discrete and Computational Geometry, 1999, 21: 405-420.

[11] M. Ramanathan and B. Gurumoorthy. Constructing medial axis transform of extruded and revolved 3D objects with free-form boundaries. Computer-Aided Design, 37(13), 2005.

[12] J.J. Chou. Voronoi diagrams for planar shapes. IEEE Computer graphics and application. 1995: 52-59.

[13] D. Lavender, A. Bowyer, J. Davenport, *et al*. Voronoi diagrams of set-theoretic solid models. IEEE Computer Graphics and Applications, 1992, 12(5): 69-77.

[14] A. Meijster, J. B. T. M. Roerdink, W. H. A Hesselink. General algorithm for computing distance transforms in linear time. In: Mathematical Morphology and its Applications to Image and Signal Processing, Kluwer Acad. Publ., 2000: 331-340.

[15] T. Hirata. A unified linear-time algorithm for computing distance maps. Information Processing Letter, 1996, 58(3):129-133.

[16] T.K. Dey, H. Woo and W. Zhao. Approximate MA for CAD models. In Proc. of Solid and Physical Modeling 2003, Seattle, Washington, USA. Jun. 16-20, 2003.

[17] G. Smogavec and B. Zalik. A fast algorithm for constructing approximate medial axis of polygons using Steiner points. Advances in engineering software, 2012, 52: 1-9.

[18] J. Giesen; B. Miklos, M. Pauly. The medial axis of the union of inner Voronoi balls in the plane. Computational geometry theory and applications 45(9), 2012.

[19] B. Miklos, J. Giesen, M. Pauly. Discrete scale axis representations for 3D geometry. In Proc. ACM SIGGRAPH, pages 394–493, 2010.

[20] J. Kustra, A. Jalba, A. Telea. Surface and curve skeletonization of large 3D models on the GPU. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(6): 1495-1508.

[21] D.Reniers, J. J Van Wijk., A. Telea. Computing Multiscale Curve and Surface Skeletons of Genus 0 Shapes Using a Global Importance Measure. IEEE TVCG, 2008, 14(2): 355-368.

[22] G.D. Rong, T. S. Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. Proc. ACM Symposium on Interactive 3D Graphics and Games (I3D), 2006, 109-116.

[23] H.S. Zhu, Y.S. Liu, *et al*. Constructive generation of the medial axis for solid models. Computer-Aided Design, 2015, 62:98-111.

[24] A. S. Glassner. An introduction to ray tracing. Academic press Ltd., London, UK, 1989.